第7章 函数重载(overload)与默认参数

本章内容

- 如何方便地使用函数名
- 通过重载，使得C++中的一个函数能够根据参数的不同，代表不同的含义。

overloading重载

- Overloading based on scopes
- Overloading based on type signatures

名字修饰

- 函数的名字修饰（name decoration）是编译器在编译期间创建的一个字符串，用来指明函数的定义或原型。LINK程序或其他工具有时需要指定函数的名字修饰来定位函数的正确位置。当然，在某些情况下需要指定函数的名字修饰，例如在C++程序中，为了让LINK程序或其他工具能够匹配到正确的函数名字，就必须为重载函数和一些特殊的函数（如构造函数和析构函数）指定名字修饰。

名字修饰

- 下面的代码中：

```
void f();

class X
{
  public:
     void f();
};
```

class X内部函数f()不会与全局的f()发生冲突。在C++编译器内部，这两个函数被翻译成_f和_X_f

名字修饰

- 函数调用时，调用者依次把参数压栈，然后调用函数。函数被调用以后，在堆栈中取得数据，并进行计算。函数计算结束以后，或者调用者、或者函数本身修改堆栈，使堆栈恢复原装。
- 在参数传递中，有两个很重要的问题必须得到明确说明：
  - 当参数个数多于一个时，按照什么顺序把参数压入堆栈
  - 函数调用后，由谁来把堆栈恢复原状
- 在高级语言中，通过函数调用约定来说明这两个问题。常见的调用约定有：stdcall、cdecl、fastcall

名字修饰

- C编译器的函数名修饰规则
  - 对于__stdcall调用约定，编译器和链接器会在输出函数名前加上一个下划线前缀，函数名后面加上一个“@”符号和其参数的字节数，例如_functionname@number。
  - __cdecl调用约定仅在输出函数名前加上一个下划线前缀，例如_functionname。
  - __fastcall调用约定在输出函数名前加上一个“@”符号，后面也是一个“@”符号和其参数的字节数，例如@functionname@number

名字修饰

- C++编译器的函数名修饰规则
  - 更复杂，信息更充分，通过分析修饰名不仅能够知道函数的调用方式，返回值类型，参数个数甚至参数类型。Cdecl，fastcall，stdcall调用方式，函数修饰都是以一个“?”开始，后面紧跟函数的名字，再后面是参数表的开始标识和按照参数类型代号拼出的参数表。
  - 函数声明：int Function1 (char *var1,unsigned long);
  - 函数修饰名为“?Function1@@YGHPADK@Z”
    - __stdcall方式，参数表的开始标识是“@@YG”

使用不同的参数类型进行重载

- 一个类中，包含下面两个方法是合法的：
  - void print(char);
  - void print(float);
- 在编译器内部，这两个函数被翻译成
  - _print_char和_print_float

用返回值重载？

- 考虑这两个函数：
  - void f();
  - int f();
- 对于下面的语句，C++应该翻译为对哪个函数的调用？
  - f()
- 用返回值重载在C++中禁止

安全类型连接

- 第一个文件中有
  - void f(int) {};
- 第二个文件中有

```
void f(char);
int main(){
    f(1);
}
```

- C中可以编译连接成功，C++中出错。

重载示例

- C07:Stash3.h
- C07:Stash3.cpp
- C07:Stash3Test.cpp
- 重载了构造函数，两个构造函数表示不同的含义。
- 使用第一个构造函数时，没有内存分配给storage，内存分配是在第一次调用add时执行。
- 第二个构造函数预先为Stash申请了一块内存

默认参数

- 在stash3.h中，Stash有两个构造函数，其中，第一个构造函数只是第二个构造函数的一个特例：它的初始size为零。
  - Stash(int size); //zero quantity
  - Stash(int size, int initQuantity);
- C++通过默认参数，提供了一种简化的办法，用一个函数声明来代替：
  - Stash(int size, int initQuantity=0);
- 下面两种定义对象的效果是相同的：
  - Stash A(100), B(100,0);

默认参数

- 默认参数的语法与使用：
  - （1）在函数声明或定义时，直接对参数赋值。这就是默认参数；
  - （2）在函数调用时，省略部分或全部参数。这时可以用默认参数来代替。

默认参数

- 只有参数列表后部的参数才是可以默认的，也就是说，不可以在一个默认参数后面又跟一个非默认的参数
- 一旦在一个参数列表中开始使用默认参数，那么这个参数后面的所有参数都必须是默认的。
- 默认参数只能放在函数声明中，通常定义在一个头文件中。有时人们为了阅读方便，在函数定义头上放一些默认的注释：

```
void fn(int x /* =0 */){……}
```

默认参数

- 占位符（placeholder）参数
  - 允许把一个参数用做占位符而不去用它，函数声明：
  - void f(int x, int =0, float=1.1);
  - void f(int x, int, float flt){/***/};
- 方便改动
  - 以后可以修改函数定义而不需要修改所有的函数调用
  - 用一个有名字的参数也可以实现，但是编译器会报
  - 如果开始用了一个函数参数，后面需要删除它，不需要改动调用该函数的代码

默认参数

- 当重载函数与默认参数共同使用时，避免参数重叠导致的二义性问题
- Test类，有一个构造函数Test()，还有一个构造函数Test(int a=0,int b=0)，那么执行Test()时候程序会调用哪一个？

默认参数

- 声明：
  - func(int);
  - func(int, int =4);
  - func(int a=3, int b=4, int c=6);
  - func(float a=3.0, float b=4.0 float c=5.0);
  - func(float a=3.0, float b=4.0 float c=5.0 float d=7.9);
- 调用
  - func(2);　　//可调用前3个函数，出现二义性
  - func(2.0);　　//可调用后2个函数，出现二义性

选择重载还是默认参数

- 自动内存管理模块：
  - C07:Mem.h
  - C07:Mem.cpp
  - C07:MemTest.cpp

选择重载还是默认参数

- 问题：使用默认参数的问题？
  - MyString(char* str=“”);
- 如果想让上述代码的效率不降低，代码需改为如下的形式：

```
MyString::MyString(char* str)
{
        if(!*str){
                buf = 0;
                return;
        }
        buf = new Mem(strlen(str)+1);
        strcpy((char*)buf->pointer(),str);
 }
```

小结

- 不能把默认参数作为一个标志去确定执行函数的哪部分。在这种情况下，只要能够，就应该把函数分解成两个或多个重载的函数。
- 一个默认的参数值应是一个在一般情况下该参数的值，这个值出现的可能性应该比其他值大。
- 默认参数的一个重要应用是在开始定义时用了一组参数，而使用一段时间后发现要增加一些参数。通过把这些新增参数作为默认参数，可以保证使用这一函数的客户代码不会受到影响。
- 一般不同时使用构造函数的重载和有默认参数的构造函数